(3)

(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2000－112758
(P2000－112758A)
(43)公開日　平成12年4月21日(2000.4.21)

(51)Int.Cl.'		識別記号	FI		テーマコード'(参考)
G06F	9/38	330	G06F	9/38	330K
		310			310E
		370			370A
	9/46	360		9/46	360B

審査請求　未請求　請求項の数1　OL　(全10頁)

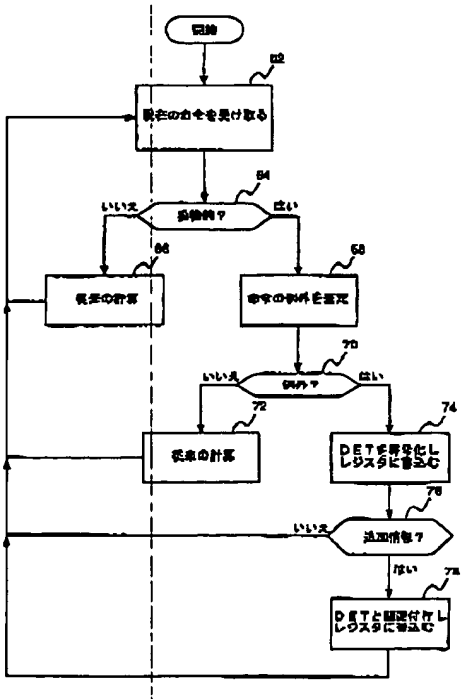
(21)出願番号　特願平11－267022
(22)出願日　平成11年9月21日(1999.9.21)
(31)優先権主張番号　164327
(32)優先日　平成10年10月1日(1998.10.1)
(33)優先権主張国　米国(US)

(71)出願人　398038580
ヒューレット・パッカード・カンパニー
HEWLETT－PACKARD COMPANY
アメリカ合衆国カリフォルニア州パロアルト　ハノーバー・ストリート　3000
(72)発明者　ガウタム・ビー・ドシ
アメリカ合衆国94086カリフォルニア州サニーベイル、マデラ・アベニュー　442、ナンバー　10
(74)代理人　100081721
弁理士　岡田　次生

最終頁に続く

(54)【発明の名称】　投機的実行中に発生する例外を遅延させるシステム及び方法

(57)【要約】
【課題】命令セットの投機的実行を支援する手法を提供する。
【解決手段】実行中に命令セットの命令を評価して個々の命令が投機的か非投機的かを判定するステップ（64）と、投機的命令の各々を評価してそれが例外を発生するかを判定するステップ（70）と、例外発生する各々の投機的命令について、CPUのレジスタの使われていないレジスタ値に遅延例外トークン（DET）をコード化するステップ（74）とを備える。本発明のシステムは、命令セットの命令を評価して個々の命令が投機的か非投機的かを判定する回路と、投機的命令の各々を評価してそれが例外を発生するかを判定する回路と、査定手段に応答してCPUのレジスタの使われていないレジスタ値にDETをコード化する回路とを備える。



【特許請求の範囲】
【請求項１】非投機的および投機的命令を含む中央処理装置の命令セットの投機的実行を支援する方法であって、
実行中に前記命令セットの命令を評価して、個々の命令が投機的か非投機的かを判定するステップと、
上記投機的命令の各々を評価して、その命令が例外を発生するかどうかを判定するステップと、
例外を発生する投機的命令の各々について、上記中央処理装置のレジスタの使われていないレジスタ値に遅延例外トークンをコード化するステップと、
を含むことを特徴とする方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、概ね、コンピュータ・プログラム・コードの最適化に関し、さらに詳しくは、投機的実行中に発生する例外の遅延によりコードの最適化を支援するシステム及び方法に関する。
【０００２】
【従来の技術】公知のように、コンピュータ・システムの性能は、コンピュータ・プログラムのコードを最適化して、コンピュータがより速くプログラムを実行できるようにすることによって向上できる。プログラムの最適化におけるステップの一つに、スケジューリングと呼ばれる処理がある。スケジューリングとは、プログラムを構成する一連のコンピュータ演算を編成して実行する処理である。スケジューリング処理中、プログラムの演算は、プログラムが特定のＣＰＵ設計においてより効率的に実行されるよう、配置、削除または移動される。スケジューリングには一般に２つの形式がある。ハードウェアによってプログラムの実行中に行われる動的スケジューリングと、コンパイラによって実行の前に行われる静的スケジューリングである。これらの技術のいずれか一方、又は両者の組合せを用いて、コンピュータ・プログラム内の演算をスケジューリングしてコンピュータ・システムで処理することができる。
【０００３】コンピュータ・プログラムは、コンピュータ・システム内の中央処理装置（ＣＰＵ）により実行される一連の命令から成る。プログラムは一般に、高級言語に書かれ、それからＣＰＵの命令セット・アーキテクチャと互換性を持つ一連の命令にコンパイルされる。プログラムは、しかし、コンピュータの命令セット・アーキテクチャに従って「機械語」で直接書くこともできる。命令セット・アーキテクチャは、命令内の演算子およびオペランドを含む、演算のフォーマットあるいはコード化形式を規定している。ＣＰＵの構造および用いられるスケジューリング技術によって、各命令は１つ又はそれ以上の演算を有することがある。１つの演算は、加算、減算、ロード、記憶、分岐等の機能を表す演算コードとしてコード化された演算子を含む。さらに演算は、オペランドおよび演算の結果を識別する。このため、演算は、オペランドのレジスタといった記憶位置を識別するコードを含むのが一般的である。最適化技術を用いてＣＰＵにより編成され実行されるのは、これらの演算である。
【０００４】最適化には、様々なレベルがある。最適化の一つのレベルに局所的最適化があり、直線的なコード・フラグメントあるいは「基本ブロック」内のコードがより能率的に動作するように処理される。「基本ブロック」は、分岐や分岐目標によって区切られた、分岐も分岐目標も含まない連続した命令の集合と定義される。このことは、基本ブロック内のある命令が実行された場合、その基本ブロック内の全ての命令が実行されるということ、すなわち、基本ブロック内に含まれる命令は、全てが実行されるか１つも実行されないかのどちらかであるということを意味する。基本ブロック内の命令は、その基本ブロックを目標とする先行の分岐からその基本ブロックに制御が渡った時点で実行に移される（ここでいう「目標とする」とは、実際に選択された分岐を介して明示的に目標とすることと、実際には選択されていない分岐を介して暗示的に目標とすることの両方を含む）。このことは、制御が基本ブロックに渡された場合には、基本ブロック内の全ての命令が必ず実行され、制御が基本ブロックに渡されない場合には、基本ブロック内の全ての命令は決して実行されないということを意味する。制御が渡る前に命令を実行する、またはその実行を特定する動作を「投機」と呼ぶ。プロセッサによりプログラムの実行時に行われる投機は「動的投機」と呼ばれ、コンパイラにより特定される投機は「静的投機」と呼ばれる。動的投機は、従来技術において知られている。
【０００５】ある命令が別の命令の結果を必要としない場合、この２つの命令は「独立」していると判断される。１つの命令がもう一方の命令の結果を必要とするときには、それらは「従属」命令と呼ばれる。従属命令は逐次実行しなければならないが、独立命令は並行して実行できる。プログラムの性能は、独立命令を識別して、できるだけ多くの独立命令を並行して実行することにより向上できる。経験から、個々の基本ブロック内だけを探索するよりも、複数の基本ブロックに渡って探索を行う方がより多くの独立命令を見付けられることが分かっているが、複数の基本ブロックの命令を同時に実行するには、投機を必要とするのが一般的である。独立命令を識別してスケジューリングし、それにより性能を向上することは、コンパイラ及びプロセッサの主要な任務の一つである。
【０００６】
【発明が解決しようとする課題】コンパイラ及びプロセッサの設計では、世代を追うごとに、独立命令の検索の範囲を広げる傾向にある。従来技術の命令セットでは、

例外を発生する可能性がある命令をコンパイラによって投機することはできない。その理由は、その命令がもし例外を引き起こすと、プログラムが、例外を発生してはならない時に誤って例外を発生してしまう可能性があるからである。このため、コンパイラの持つ広範な独立命令の検査適用範囲が限定されてしまい、投機はプログラム実行時にプロセッサによって動的投機により行わなければならなくなる。しかし、動的投機は、非常に複雑なハードウェアを必要とする上、その複雑さは動的投機が行われる基本ブロックの数に伴って指数的に増大し、それにより動的投機の適用範囲は事実上制限される。一方、コンパイラに可能な独立命令の探索の範囲ははるかに広く、潜在的にはプログラム全体に渡る。さらに、１つの基本ブロックの境界の全域に渡って静的投機を実行するようにコンパイラを設計してあれば、複数の基本ブロック境界を越えて静的投機を行うとしても、複雑さはごく僅かしか増大しない。

【０００７】局所的最適化技術の例は、共通式の削除および定数の伝搬である。別のレベルの最適化としては大域的最適化がある。これには、プログラム内で条件付き分岐を越えて局所的最適化技術を拡張することと、ループ最適化のための変換を行うことが含まれる。大域的最適化の一つの形式に、コードの移動がある。コード移動の例としては、ループを繰返す度に同じ値を計算するループからコードを除去することがある。第３のレベルの最適化は、機械依存性の最適化である。機械依存性の最適化は、ＣＰＵに特有の構造上の特性を利用するようにコードを処理することを含む。例えば、ＣＰＵが並列に命令を実行するためのパイプライン機能ユニットを有する場合、コードを再配列してパイプライン・パフォーマンスを向上できる。

【０００８】プログラムを最適化するため、コードは、投機的コード移動と呼ばれるスケジューリング処理において、条件付き分岐上に移動されることがある。投機的コード移動とは、ある命令をその実行を制御する条件付き分岐上へ移動することを指す。「投機的な」命令の実行は、投機的実行または先行実行とも呼ばれるが、それは、その命令が、プログラムにおいて実際に使われるかどうかが分かる前に実行されるからである。投機的コード移動では、命令レベルでの並列性を向上できる。多くの命令は長い待ち時間を有する（すなわち、実行されるまでに数クロック・サイクルかかる）ので、投機的に命令を実行することは有利である。命令が他の場合であれば引き起したであろう遅れは、前もってその命令を発行することにより、最小限に抑えることができる。投機的コード移動は、また、冗長性の除去のような他の最適化にも役立つ。

【０００９】静的投機を行おうとする場合、解決すべき問題がいくつかある。その中でも最も重要なものの一つは、静的に投機された命令が遭遇する例外条件の取り扱いである。

【００１０】上述のように、投機的命令上の例外を命令の実行の時点で引き渡すことは不可能である。従って、そこからその命令が投機された基本ブロック（「元の基本ブロック」として知られる）に制御が渡るまでは、例外の引き渡しを遅延させるためにコンパイラで見ることができる機構が必要である。動的に投機された命令における例外を遅延させ、その後で引き渡す同様の機構は、従来技術でも存在する。しかし、そうした機構は当然、コンパイラに可視ではないため、コンパイラで操作してコンパイラの指示による投機に役立てることは出来ない。従来技術においては、静的に投機された命令の例外を遅延させ、その後で引き渡す方法あるいは装置は可能ではなかった。また、従来技術においても、静的投機は限られた形式で存在するが、（１）そうした形式には、例外条件の遅延及びその後の回復が含まれず、また（２）そうした形式では、本発明の幅および範囲にあるような静的投機は実現されない。

【００１１】従来技術における限られた静的投機の別の例は、例外を引き起こさない命令の投機である。例えば、比較命令は一般に、いかなる例外も発生しないものと規定される。副作用は宛先の書込みだけであるので、適切に設計されたコンパイラは比較命令を投機することができる。もし制御が比較命令の元の基本ブロック(originating basic block)に渡らない場合には、その宛先は単に放棄される。別の例としては、コンパイルの際に有効であることが分かっており且つ実行中不変であることが分かっているアドレス、例えばグローバル変数などからのロード命令がある。こうした条件では、もし何らかの例外が起こっても、それが致命的なものではなく、副作用なしで投機的に処理できることが保証されている。但し、投機的な例外の処理により総合的な性能が低下する恐れがある。重ねて留意すべき点は、いま説明した限られた形の投機では、例外の遅延は含まれず、また可能でもなく、ある限られたクラスの命令にしか適用されないということである。

【００１２】したがって、静的投機を行おうとする場合には、投機的命令における例外を遅延させる機構を、できるだけ多くの形の投機に適用できるようにする技術が必要である。この機構は、待ち時間が非常に短いものでなければならない。さもなければ、投機を行ってコンパイルしたプログラムの性能が、実際には、投機なしでコンパイルした同じプログラムを下回る恐れがある。この機構はまた、これまでのソフトウェア資産の実行を可能にし、ソフトウェア開発者への影響を最小にし、且つソフトウェアの実行の選択範囲を最大にするために、ソフトウェアの形式及び構造に課する制限が最小限でなければならない。最も幅広い範囲のソフトウェアにおいて最大限の性能を得るために、この機構は、プログラムの挙動にコンピュータ・システムが動的に適応できるように

するという特性を有するのが望ましい。

【００１３】投機的実行中に発生する例外に対処する他の方法も知られている。一つの保守的な方法は、「安全な投機」と呼ばれている。この方法においては、例外を発生しない演算だけが投機的に移動される。この方法では、多くの演算の投機的な移動が予め排除されているため、命令レベルでの並列性は十分に向上されない。さらに、ロード演算が投機的に実行されず、従ってメモリの待ち時間を隠蔽するという利点がない。

【００１４】別法として、ブースティングと呼ばれる方法がある。この方法においては、投機的演算は、自身の基本ブロックに戻るパスを付加される。この状態情報は、例外を遅延させるため、プロセッサが異なるパスを取るか又はこの演算の結果が非投機的な演算中で使用されるまで保持されなければならない。

【００１５】この状態情報をセーブする必要があることが、ブースティング技術の欠点である。この状態情報を格納するには追加のメモリが必要となる。従って、ブースティングを実現し得る範囲と、分岐方向を格納するために必要な追加の演算コード・ビットとの間で、一定の妥協をせざるを得ない。ある演算の移動で越えることができる分岐の数は、状態情報を格納できるメモリにより制限される。

【００１６】別の方法は、例外の遅延にポイズン・ビット(poison bit)を使用することである。この方法では、例外が発生すると、プロセッサは投機的演算の結果レジスタを、ポイズン・ビットでマークする。別の投機的演算がこの演算の結果を使用するときには、プロセッサはポイズン・ビットを演算の結果レジスタにセットすることにより、この例外を伝搬できる。この例外の処理は、非投機的演算がポイズン・ビットを消費するまで遅延される。その時点でプロセッサは例外を報告または処理することができる。

【００１７】ポイズン・ビット法では、投機的演算と非投機的演算とを区別するために、命令セット・アーキテクチャ内の投機的演算の演算コードに余分なビットを付加する必要があるのが一般的である。そのため、命令セットの複雑さが増大すると共に、レジスタ・ファイル内に追加のメモリが必要となるという欠点がある。さらに、ファンクション・コールまたは文脈切替えにおいてレジスタをこぼれる際には、ポイズン・ビットを記憶させなければならない。例えば６４ビットのデータを保持するレジスタは６５ビットのメモリにこぼれる必要があるから、ポイズン・ビットを記憶させることは難しい。

【００１８】さらに別の方法は、タグ付けと呼ばれる。この方法では、各演算は、関連付けられたタグを有する。一般に、ゼロのタグはその演算が非投機的であることを表す。投機的演算においては、タグは遅延された例外に関する情報を格納するタグ・テーブルのようなプロセッサ内のメモリを指す。この方式では、演算のホーム・ブロックにコミット演算を挿入して、遅延した例外を検査する。

【００１９】タグ付け法による１つの問題は、投機の量が一般に、タグ付けできる演算コードの数により制限されるということである。より多くのビットがタグのコード化に必要になると、使用可能なビットが減り、命令セット・アーキテクチャの演算のレパートリが低下する。別の問題は、分岐方向によりコミット演算がスキップされた場合に、タグに格納されている情報を明示的に消去する必要があることである。

【００２０】それゆえ、投機的実行中に発生する例外を遅延させるシステム及び方法であって、従来技術の短所を克服するシステム及び方法が望まれている。

【００２１】

【課題を解決するための手段】本発明の目的、利点および新しい特徴のいくつかは、一部にはこのあとに続く説明に記載され、一部には、当業者においては、以下を検討することにより明らかになるか又は本発明の実施面から理解されるであろう。本発明の目的および利点は、特に添付の請求項において示される手段及び組合せにより実現され達成される。

【００２２】上記利点および新しい特徴を達成するため、本発明は、概ね、非投機的および投機的命令を含む中央処理装置（ＣＰＵ：Central Processing Unit）の命令セットの投機的実行を支援するためのシステム及び方法を目指すものである。一態様では、本発明は、実行中に命令セットの命令を評価して個々の命令が投機的か非投機的かを判定するステップと、投機的命令の各々を査定してそれが例外を発生するかどうかを判定するステップとを備えた方法である。例外を発生する各々の投機的命令について、本方法は、ＣＰＵのレジスタの使われていないレジスタ値に遅延例外トークン（ＤＥＴ：Deferred Exception Token）をコード化する。

【００２３】別の態様では、本発明は、命令セットの命令を評価して個々の命令が投機的か非投機的かを判定する手段を備えたシステムを提供する。本システムは、さらに、投機的命令の各々を査定してそれが例外を発生するかどうかを判定する手段を備える。最後に、本システムは、査定手段に応動して、ＣＰＵのレジスタの使われていないレジスタ値に遅延例外トークン（ＤＥＴ）をコード化する手段をさらに備える。

【００２４】本発明の好適な実施例によれば、浮動小数点レジスタがＤＥＴを格納するために用いられる。ここでは、浮動小数点レジスタの指数部分および仮数部分にまたがる使われていないビット・シーケンスが、固有のＤＥＴ値のコード化に用られる。

【００２５】しかし、発明の思想の範囲内で、ＤＥＴに関係する追加の情報をＤＥＴに関連付けてコード化し格納することができる。ここでは、浮動小数点レジスタを、コード化されたＤＥＴ値を格納するために用いるこ

とができる。ＤＥＴ値は、一般に浮動小数点数の指数値デリミタのために予約されている複数のビットに格納されるのが好ましい。しかし、浮動小数点レジスタにおいて、予約されているか又は他で使われていないビット・コンビネーションがレジスタの指数部分に複数存在する場合には、他で使われていないビット・シーケンス内で固有のＤＥＴ値をコード化して、浮動小数点レジスタの指数部分に書込んでもよい。特定のＤＥＴ値と関連付けることができる追加情報に関していうと、この追加情報は、一般に仮数情報のために予約されている浮動小数点レジスタの部分にコード化して書き込んでもよい。この追加情報は、ＤＥＴ値と同じレジスタに書込むことでそのＤＥＴ値と関連付けられる。

【００２６】本発明の別の実施例によると、本発明の思想及びその教えるところを逸脱せずに、浮動小数点レジスタ以外のレジスタを用いることもできる。本発明の目的上重要なことは、所定のレジスタにビットを追加する必要がないということである。その代わりに、予約されている値または他で使われていない値を利用して、コード化されたＤＥＴ値を格納する。さらに、ＤＥＴ値を、他で使われていないビット・コンビネーションを有する複数のレジスタ内にコード化し格納してもよい。

【００２７】デコーダを用いてコード化されたＤＥＴ値を識別して、実行、回復または他の適切な動作を行ってもよい。

【００２８】

【発明の実施の形態】次に図面を参照して本発明の実施形態を説明するが、本発明はここで開示される実施の形態に限定されるものではなく、あらゆる代案、修正および等価物は、添付の請求項に記載の本発明の精神および範囲に含まれるものである。

【００２９】概要として、図１に、本発明の実施例を実行できるコンピュータ・システム２０の概略ブロック図を示す。コンピュータ・システム２０は、システム・バス２８を介してメモリ２４及び１つ以上の周辺装置２６に接続されたＣＰＵ２２を備える。システム・バス２８は、データ及び制御信号をＣＰＵ２２、メモリ２４及び周辺装置２６へ伝送する。メモリ２４は、好ましくはランダム・アクセス・メモリ（ＲＡＭ：Random Access Memory）から成るが、リード・オンリ・メモリ（ＲＯＭ：Read Only Memory）又はＲＡＭとＲＯＭとの組合せにより実装してもよい。メモリ２４は、コンピュータ・システム２０において実行できる１つ又はそれ以上のプログラムのデータを格納する。

【００３０】図２は、本発明の実施例におけるＣＰＵ（またはプロセッサ）２２の概略ブロック図である。プロセッサ２２は、複数の機能ユニット３０と、１つ又はそれ以上のレジスタ・ファイル３２と、命令ユニット３４とを備える。レジスタ・ファイル３２は、値、アドレス及び場合によっては他のデータを格納するいくつかの汎用レジスタ３６を含むのが一般的である。用語「汎用レジスタ」には、例えば、浮動小数点レジスタ、固定小数点レジスタ及び述語レジスタ等が含まれる。

【００３１】当業者には明らかなように、プロセッサ２２のアーキテクチャは、本発明の思想およびその教えるところを逸脱せずに変更することができる。図示のアーキテクチャ自体は、単に可能な一実施例におけるプロセッサ２２の高水準でのハードウェア設計を表現したものである。本発明により実行される投機的実行は、特に複数の機能ユニットを有する特殊なＣＰＵまたは複数のパイプライン機能ユニットを有するＣＰＵを含む、さまざまな設計のＣＰＵの性能を向上できる。なお、複数の機能ユニットを有するＣＰＵは、命令シーケンスの並列実行が可能である。ここで説明するように、ある特定のプロセッサ２２用のコンパイラは、コードをコンパイルしてプロセッサ上で実行する際にこの並列処理を利用する。この並列処理の一部に投機的実行の処理があり、投機的実行は特に超長命令語（ＶＬＩＷ：Very Long Instruction Word）コンピュータ内の性能の向上に効果的である。

【００３２】プログラムの実行処理において、ＣＰＵ２２は、メモリ２４に格納された一連の命令を実行する。命令ユニット３４は、システム・バス２８を介してメモリから命令を取り出し、その命令を解読する。ＣＰＵの種類および／又は用いられるスケジューリング方法によって、命令は複数の演算を有する場合がある。命令ユニット３４は、機能ユニット３０又は複数の機能ユニット（図２に、ボックスの積み重ねとして示す）に演算を発行する。命令ユニット３４は、命令内の演算を実行するように機能ユニット３０に制御信号を送る。これらの制御信号に応じて、機能ユニット３０は、レジスタ・ファイル３２内の該当するレジスタからアドレスまたは値などのデータを読み込んで、演算を実行する。演算によっては、機能ユニット３０がレジスタ・ファイル３２に結果を書き込む場合もある。例えば、メモリ・ストア演算では、機能ユニット３０はレジスタ・ファイル３２に格納されたメモリ・アドレス及び値を読み込み、その値をメモリ２４へ直接転送する。

【００３３】ＣＰＵ２２の具体的な構造は変化することがあるものの、本発明は、演算の実行を制御するために述語を使用するプロセッサにおける投機的実行を支援する。

【００３４】本発明によれば、レジスタ・ファイル３２の１つ又は複数のレジスタを利用して、例外を発生する投機的命令に対して「遅延例外トークン」（ＤＥＴ）を命名する。ここで、本発明は、１つ又はそれ以上のレジスタの他で使われていないビット・シーケンスをコード化してＤＥＴを命名する。これにより、従来技術のシステムにおいてなされたようにＤＥＴを識別するレジスタに余分なビットが付加されること（例えば、「タグ付

き」レジスタや「ポイズン・ビット」など）が避けられるため有利である。さらに後述するように、好適な実施例においては、浮動小数点レジスタがＤＥＴをコード化するために利用される。

【００３５】さらに解説及び説明すると、命令は二つのクラス、すなわち投機的な命令と非投機的な命令とに分けられる。まず最初に、全ての命令は非投機的であるとマークされる。コンパイラは、ある命令をその命令の基本ブロックの外側でスケジューリングした場合、その命令を投機的であるとマークする。非投機的命令は、例外条件に遭遇と例外を発生する。投機的命令は、例外条件に遭遇しても例外を発生せず、代わりにその宛先にＤＥＴを書き込む。ＤＥＴはこの時点では、宛先が正しい結果を有していないという単なる記号に過ぎない。非投機的命令は、遅延例外トークンを読み込むと、例外を発生する。投機的命令は、ＤＥＴを読み込むと、その命令の宛先（ここでも、宛先は正しい結果を有していない）にＤＥＴを書き込む。この動作は「伝搬」と呼ばれる。非投機的命令を所定の投機的命令の元の基本ブロック内に配置し、且つ投機的命令の宛先（またはＤＥＴが伝搬する可能性のあるいずれかの位置）を読み込むようにその非投機的命令を構成することによって、投機的命令で生成されたＤＥＴを、元の基本ブロックに制御が渡る時点で例外に変換することが出来る。ＤＥＴを例外に変換し且つ例外条件を修正した後には、以前生成したＤＥＴを全て、正しい結果に置換する必要がある。これは、「回復」と呼ばれる処理により達成される。回復では、コンパイラによって、生成した追加のコードでプログラムを増補する必要がある。システムは、例えばプログラム・サイズを最小にするために回復コードを含まないことを選んでもよいが、その場合、例外を遅延させる機会は非常に限定される。なお、本発明は回復の方法を狙いとしているものではないため、この件についてはこれ以上の説明を行わない。

【００３６】投機的命令が例外条件に遭遇する度毎にオペレーティング・システム（ＯＳ：Operating System）に対して例外を生成し、ＯＳに例外条件を修正させる（その修正にプログラム管理下の副作用が伴わない場合）か手作業で命令の宛先にＤＥＴを書き込むことで例外を遅延させることも考えられる。この方法の欠点は、ＯＳへの例外の生成が待ち時間の大きい演算であって、一般的にプロセッサ・パイプラインのフラッシュやキャッシュ・ミスを引き起こすというである。さらに、ＯＳは、アドレスのポストインクリメント等といったソフトウェア内の命令の補助演算を全てエミュレートすることも要求されるため、総合的な待ち時間がさらに長くなる。もし、この待ち時間の長い演算が、あらゆる投機的命令のあらゆる例外条件において起こるとすれば、投機を伴うプログラムの性能は投機なしの同じプログラムの性能の下をはるかに下回ることであろう。

【００３７】１つのプログラムは、複数の「コンパイル単位」または「モジュール」から成るのが一般的である。多くの場合、全てのモジュールが同時に又は同じコンパイラによってコンパイルされるというわけではない。さらに、「ダイナミック・リンク」として知られる処理により、ある種のモジュールは実行時にだけ認識され、従ってコンパイル時には分からないということが可能である。モジュールの共有は、ソフトウェア開発において一般的な方法であって（例えば、ライブラリ）、異なるモジュールを異なった程度の回復コード（例えば、全ての投機的命令を回復する、一部の投機的命令を回復する、又は全く回復しない）でコンパイルすることができる。

【００３８】本発明の説明に合わせて、図３を参照する。図３は、ＤＥＴを格納するための機構の概略ブロック図を示す。本発明の重要な態様は、他で十分に利用されていないレジスタを利用してＤＥＴ値を格納することである。ここでは、レジスタ・ファイル３２の１つ又はそれ以上のレジスタが、予約されているか又は使用されていないビット・シーケンスを有している分だけ十分に利用されていない。本発明は、これらの使われていないビット・シーケンスのいくつかを利用する。本発明では、ＤＥＴ値を、レジスタ・ファイル３２内の１つ又はそれ以上のレジスタの予約されているか又は使われていないビット・シーケンスに対応するビット・シーケンス内にコード化する。動作としては、機能ユニット３０は、例外を発生する投機的命令に遭遇するとその命令をエンコーダ４４に渡し、エンコーダ４４は、レジスタ・ファイル３２のレジスタに格納できるようなＤＥＴ値を生成する。デコーダ４６を利用して、レジスタ・ファイル３２内に格納されるＤＥＴ値を解読してもよい。

【００３９】本発明の利点の一つは、他で使われていないレジスタ値をＤＥＴ値に利用することにより、ＣＰＵにハードウェアを追加する必要がないということである。なお、従来技術のシステムでは、追加のゲート（通常、既存のレジスタに加える余分なビット）の形で追加のハードウェアを付け加えていた。

【００４０】一実施例として、発明の思想を逸脱せずに、複数のレジスタを利用して多様なＤＥＴ値を格納してもよい。例えば、図４は、図３と同様のシステムの一例を示すが、この例では、ＤＥＴエンコーダ４４及びＤＥＴデコーダ４６が、レジスタ・ファイル３２を構成する複数のレジスタ５２、５４と連通している。

【００４１】比較として、本願の出願人に譲渡された米国特許第５，７４８，９３６号では、ＤＥＴ用に、レジスタ・ファイル内に追加のビットを設けていた。その特許には、各ＤＥＴ毎の追加ビットが開示されている。また、投機的演算で発生した例外の遅延に用いられる状態情報を格納する投機的索引テーブル（ＳＬＡＴ：Speculative Look Aside Table）が開示されている。これとは

対照的に、本発明では追加のビットをレジスタに加える必要がなく、またＳＬＡＴを付加する必要もない。その代わりに、本発明は、既存のレジスタ内の予約されているか又は他で使われていない値を利用して、状態情報を格納している。好適な実施例において、本発明は、浮動小数点レジスタ１５２をこの目的に使用する。

【００４２】浮動小数点レジスタ１５２は、図５Ａ、５Ｂに示す形をとることができる。浮動小数点レジスタ１５２（または他の選ばれたレジスタ）のビット構造は、異なるＣＰＵアーキテクチャでは相違してよいが、本発明の好適な実施例では図５Ｂの構造を利用する。ここで浮動小数点レジスタ１５２は、８２ビットのレジスタであり、１ビットの符号部と、１７ビットの指数部と、６４ビットの仮数部とを有する。好適な実施例のＣＰＵにおいては、浮動小数点レジスタ１５２の１７ビットの指数部および６４ビットの仮数部が全体として十分に活用されておらず、その中の多くのビット・シーケンスが利用されていなかった。それゆえ、これらの使われていないシーケンスの一つを、固有のＤＥＴ値を格納するために利用した。

【００４３】多くの例では、追加の情報をＤＥＴ値に関連付けることが望ましい場合がある。この種の追加情報は、本発明の思想を逸脱せずに、コード化しＤＥＴ値とともに格納できる。あるいは、別個にコード化してＤＥＴ値と関連付けてもよい（例えば参照テーブルによって）。さらにまた、例えばレジスタの限定可能なある部分（例えば指数部）が十分に利用されていないような場合には、残っている部分を利用して関連する追加情報の値を格納してもよい。

【００４４】本発明の好適な実施例において利用される浮動小数点レジスタ１５２は、レジスタ内の使われていないビット・シーケンスの使用可能度のために選ばれたことは理解されよう。しかし、本発明の思想およびその教えるところに一致したままで、他のレジスタを同様に利用することもできる。本発明の目的上重要なのは、ＤＥＴ値でのコード化に使用可能なビット・シーケンスを有する、ＣＰＵのレジスタ・ファイル３２内の１つ又はそれ以上のレジスタを利用することである。

【００４５】次に、図６を参照する。図６は、本発明により構成されるシステムにおける最高位の機能動作を示すフローチャートである。図示のように、動作は、システムがプログラムから評価すべき命令を受け取ることから始まる（ステップ６２）。各命令が評価され、それが投機的命令である（すなわち、現在の基本ブロックの外側にある）か否かが判定される（ステップ６４）。投機的でない場合には、システムは従来通りに処理を続行する（ステップ６６）。一方、命令が投機的命令であると評価された場合、システムはその命令の査定を行い、例外を発生するかどうかを判定する（ステップ６８及び７０）。

【００４６】その命令が例外を発生しない場合、システムは従来通りに処理を続行する（ステップ７２）。一方、例外が発生する場合、システムは命令の遅延を示すＤＥＴを生成する。続いて、このＤＥＴはコード化され、既存のＣＰＵのレジスタ内の予約されているか他で使われていないビット・シーケンスに書き込まれる（ステップ７４）。次に、システムは、ＤＥＴと関連付けすべき追加情報が存在するかどうかを判定する（ステップ７６）。存在する場合、その追加情報は必要に応じてコード化され、使用可能なレジスタ・スペースに書き込まれる。本発明の好適な実施例によれば、ＤＥＴ値は、浮動小数点レジスタ１５２の指数部及び仮数部の他で使われていないビット・シーケンスを用いて、浮動小数点レジスタに書き込まれる。

【００４７】上述の説明は例示および説明のために行われたものであって、本発明を網羅するものでも本発明を開示された明確な形式に限定するものでもない。上述の教えるところに鑑みて明らかな修正や変更が可能である。検討した実施例は、本発明の原理およびその実地の用途を最も良く例示するよう選択され説明されたものであり、それにより考えられる個々の使用に適した種々の実施形態および種々の修正をもって当業者が本発明を利用することを可能にするものである。

【００４８】この発明は、例として次の実施形態を含んでいる。

１．非投機的および投機的命令を含む中央処理装置（ＣＰＵ：Central Processing Unit）の命令セットの投機的実行を支援する方法であって、実行中に前記命令セットの命令を評価して、個々の命令が投機的か非投機的かを判定するステップと、投機的命令の各々を査定して、その命令が例外を発生するかどうかを判定するステップと、例外を発生する投機的命令の各々について、ＣＰＵのレジスタの使われていないレジスタ値内に遅延例外トークン（ＤＥＴ：Deferred Exception Token）をコード化するステップとを含むことを特徴とする方法。

【００４９】２．上記１に記載の方法において、前記ＤＥＴ値と関連付けられた追加情報をコード化し、この追加情報を前記レジスタに書き込むステップをさらに備えることを特徴とする方法。

【００５０】３．上記１に記載の方法において、前記レジスタが、浮動小数点レジスタであることを特徴とする方法。

【００５１】４．上記１に記載の方法において、複数のレジスタが、コード化されたＤＥＴ値を格納するために使われることを特徴とする方法。

【００５２】５．上記３に記載の方法において、前記浮動小数点レジスタが、指数情報を格納するための複数のビットと、仮数情報を格納するための複数のビットとを含むことを特徴とする方法。

【００５３】６．上記５に記載の方法において、コード

化されたＤＥＴ値は使われていない指数値であり、指数情報を格納するために使われる複数のビットに格納されることを特徴とする方法。

【００５４】７．上記５に記載の方法において、コード化されたＤＥＴ値は、指数ビットと仮数ビットとにまたがる使われていないビット・シーケンス内に書き込まれることを特徴とする方法。

【００５５】８．上記６に記載の方法において、前記ＤＥＴ値と関連付けられた追加情報をコード化し、この追加情報を仮数情報を格納するための複数のビットに書き込むステップをさらに備えることを特徴とする方法。

【００５６】９．非投機的および投機的命令を含む中央処理装置（ＣＰＵ）の命令セットの投機的実行を支援するシステムであって、前記命令セットの命令を評価して、個々の命令が投機的か非投機的かを判定する手段と、投機的命令の各々を査定して、その命令が例外を発生するかどうかを判定する手段と、前記査定を行う手段に応動して、ＣＰＵのレジスタの使われていないレジスタ値内に遅延例外トークン（ＤＥＴ）をコード化する手段とを備えることを特徴とするシステム。

【００５７】１０．上記９に記載のシステムにおいて、前記ＤＥＴ値に関連付けられた追加情報をコード化し、この追加情報を前記レジスタに書き込む手段をさらに備えることを特徴とするシステム。

【００５８】１１．上記９に記載のシステムにおいて、前記レジスタが、浮動小数点レジスタであることを特徴とするシステム。

【００５９】１２．上記９に記載のシステムにおいて、複数のレジスタが、コード化されたＤＥＴ値を格納するために使われることを特徴とするシステム。

【００６０】１３．上記１１に記載のシステムにおいて、前記浮動小数点レジスタが、指数情報を格納するための複数のビットと、仮数情報を格納するための複数のビットとを含むことを特徴とするシステム。

【００６１】１４．上記１３に記載のシステムにおいて、コード化されたＤＥＴ値は使われていない指数値であり、指数情報を格納するために使われる複数のビットに格納されることを特徴とするシステム。

【００６２】１５．上記１３に記載のシステムにおいて、コード化されたＤＥＴ値が、浮動小数点レジスタの指数情報部と仮数情報部とから成る使われていないビット・シーケンスであることを特徴とするシステム。

【００６３】１６．上記１４に記載のシステムにおいて、前記ＤＥＴ値と関連付けられた追加情報をコード化し、この追加情報を仮数情報を格納するための複数のビットに書き込む手段をさらに備えることを特徴とするシステム。

【００６４】１７．上記９に記載のシステムにおいて、前記コード化を行う手段が、前記レジスタと電気的に連通した電子回路を備えることを特徴とするシステム。

【００６５】

【発明の効果】本発明によると、投機的実行中に発生する例外を遅延させるシステム及び方法であって、従来技術の短所を克服するシステム及び方法を提供することができる。

【図面の簡単な説明】

【図１】図１は、本発明を実行できる、従来技術において公知のコンピュータ・システムのブロック図である、

【図２】図２は、本発明を実行できるプロセッサのブロック図である。

【図３】図３は、本発明により構成されるシステムを示すブロック図である。

【図４】図４は、複数の汎用レジスタとＤＥＴエンコーダ/デコーダ対との間の相互接続を示すブロック図である。

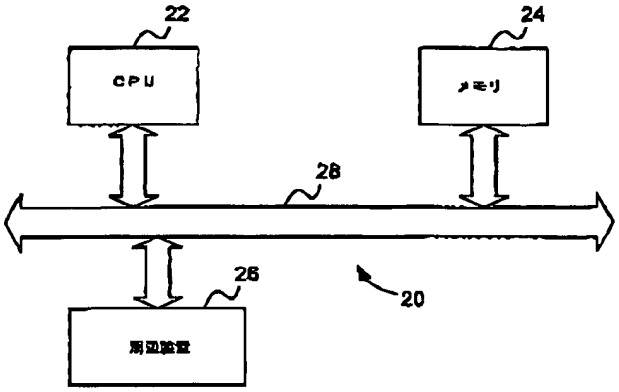
【図５】図５Ａは、好適な実施形態の浮動小数点レジスタとのＤＥＴエンコーダ/デコーダ対の相互接続を示すブロック図であり、図５Ｂは、好適な実施形態の浮動小数点レジスタのビット分割を示す線図である。

【図６】図６は、本発明によるシステムの最高位の動作を示すフローチャートである。

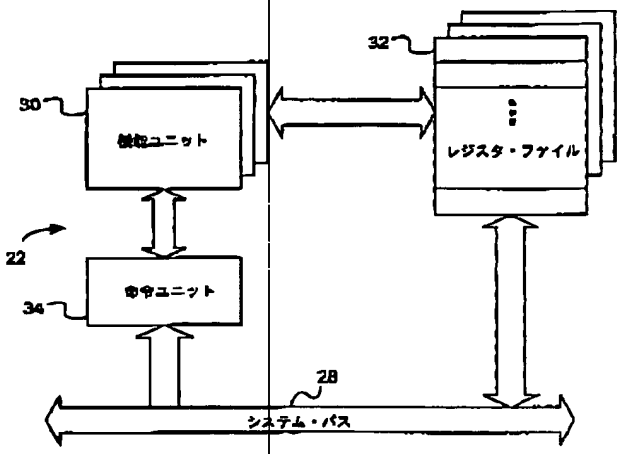
【符号の説明】

２２　ＣＰＵ　２４　メモリ
２６　周辺装置　２８　システム・バス
３０　機能ユニット　３２　レジスタ・ファイル
３４　命令ユニット　４４　ＤＥＴエンコーダ
４６　ＤＥＴデコーダ　５２、５４　汎用レジスタ１
１５２　浮動小数点レジスタ

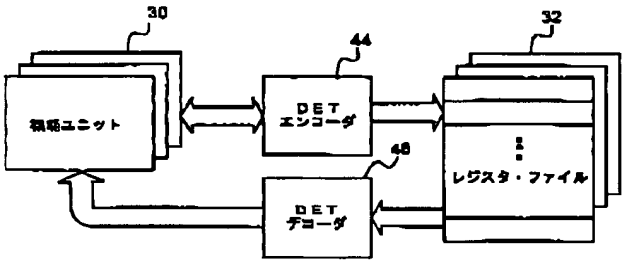
【図1】


22
CPU
24
メモリ
28
26
周辺装置
20

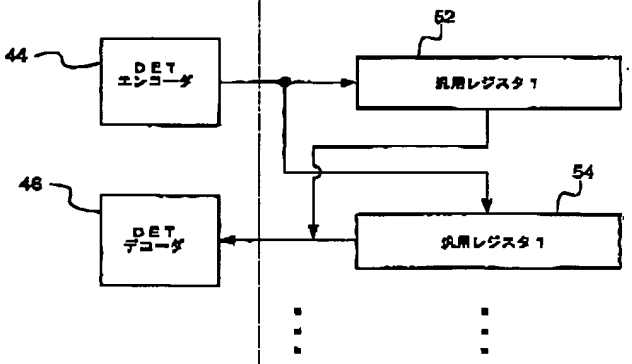
【図2】


32
30
演算ユニット
レジスタ・ファイル
22
命令ユニット
34
28
システム・バス

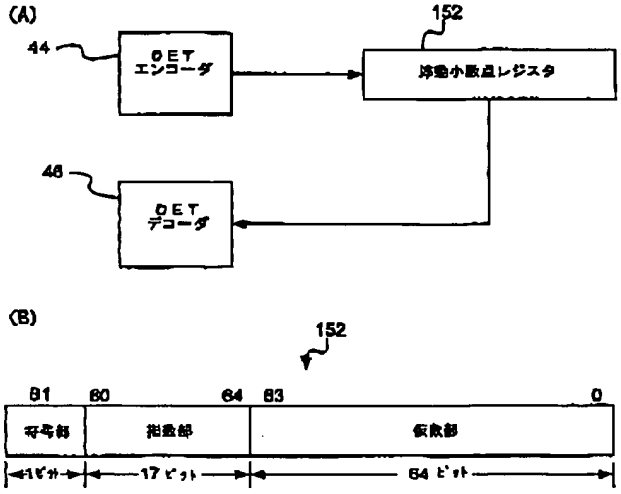
【図3】


30
44
32
演算ユニット
DET
エンコーダ
レジスタ・ファイル
46
DET
デコーダ

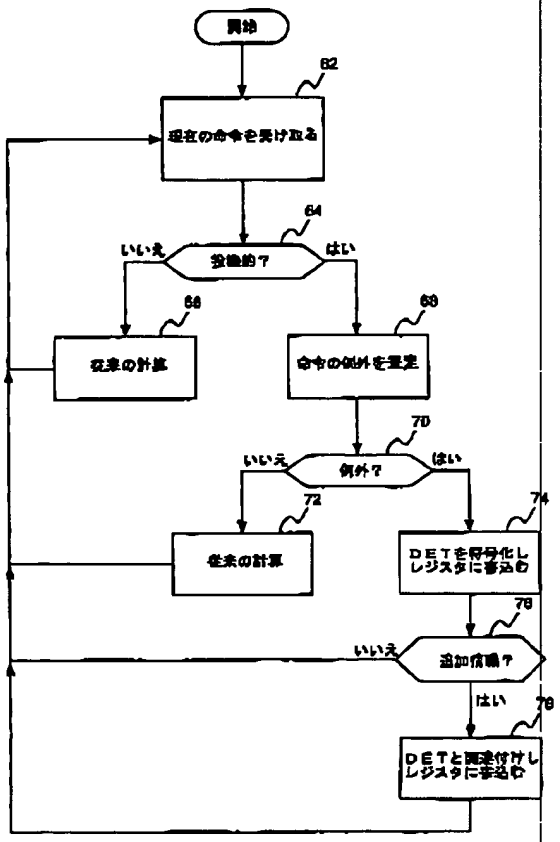
【図4】


44
DET
エンコーダ
52
汎用レジスタ1
54
46
DET
デコーダ
汎用レジスタ1

【図5】


(A)
44
DET
エンコーダ
152
浮動小数点レジスタ
46
DET
デコーダ
(B)
152
81
80
64
63
0
符号部
指数部
仮数部
1ビット
17ビット
64ビット

【図６】



フロントページの続き

(72)発明者 ピーター・マークステイン
アメリカ合衆国94062カリフォルニア州ウッドサイド、レッドランド・ロード 160、エスアールエックス 96
(72)発明者 アラン・エイチ・カープ
アメリカ合衆国94306カリフォルニア州パロ・アルト、イリマ・コート 837
(72)発明者 ジェローム・シー・ハック
アメリカ合衆国94303カリフォルニア州パロ・アルト、タリスマン・ドライブ 851
(72)発明者 グレン・ティー・コロンーボネット
アメリカ合衆国80526コロラド州フォート・コリンズ、アントラーズ・コート 4303
(72)発明者 マイケル・モリソン
アメリカ合衆国94086カリフォルニア州サニーベイル、コーンフラワー・コート 1072